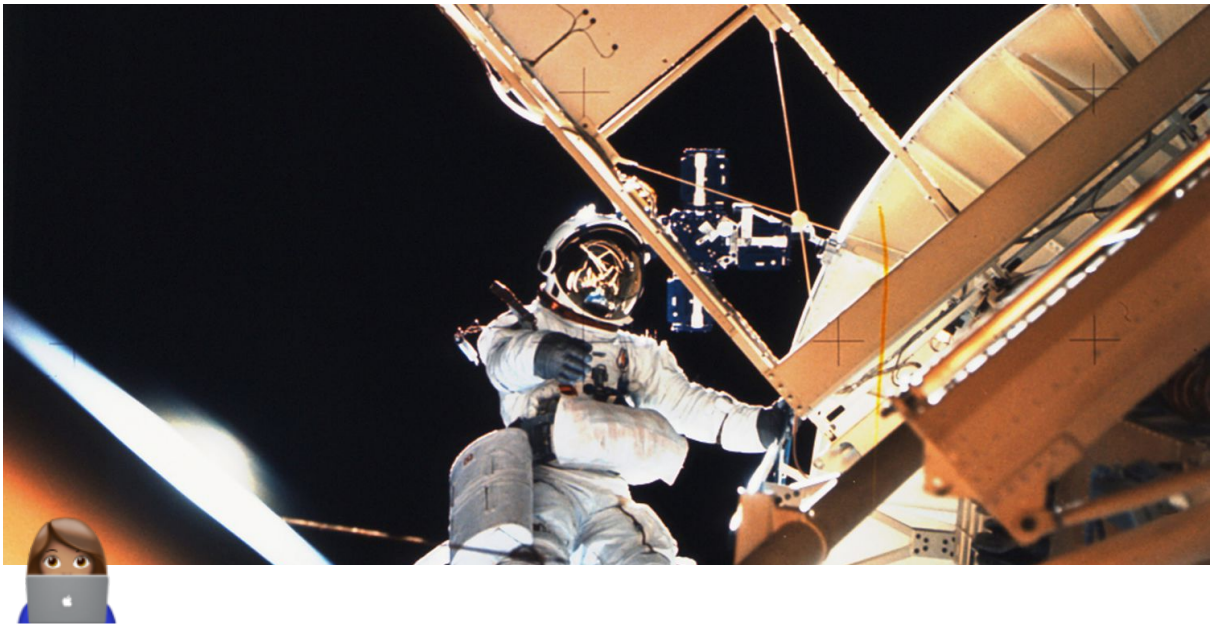

Simple Exam Registry System Using TDD (Test-Driven Development)

Project Overview:

- The system will allow registering students, enrolling them in exams, and recording their exam scores. It will ensure correct handling of edge cases (e.g., duplicate registration, invalid scores) and exception management.
- Students will be guided through writing unit tests and implementing functionality in small, testable steps.

▼ Step 1: Cloning the Repository

1. Fork the starter repository from GitHub:
    - GitHub link to the repository: (link will be provided)
    - Clone the repository locally using the command:

```
git clone https://github.com/eiramlan/student-exam
-registry.git
cd student-exam-registry
```

2. Open the project in your preferred IDE (e.g., IntelliJ IDEA, Eclipse) and familiarise yourself with the provided skeleton code, which includes:
    - `Student.java`: Skeleton class representing a student.
    - `Exam.java`: Skeleton class representing an exam.
    - `StudentRegistry.java`: Interface where you will implement registry operations.
    - `ExamRegistry.java`: Interface where you will implement exam registration and scores.

▼ Step 2: Define Requirements and Features

The following are the main features that will be implemented through TDD:

1. **Registering Students:**
    - Add a student to the registry.
    - Prevent registering the same student twice (by ID).
2. **Enrolling Students in Exams:**
    - Enrol students into exams.
    - Prevent enrolling students into the same exam multiple times.
3. **Recording Exam Scores:**
    - Record scores for exams.
    - Ensure that scores are valid (e.g., within a range of 0–100).
    - Retrieve students' scores and compute averages.

▼ Step 3: Implementing TDD for Student Registration

1. **Write the First Test:**
    - Create a test to ensure that a student can be registered.

```
@Test
public void shouldRegisterStudent() {
    StudentRegistry registry = new StudentRegistry();
    Student student = new Student("1", "John Doe");

    registry.register(student);
```

```
    assertTrue(registry.isStudentRegistered("1"));
}
```

2. **Run the Test:**
   - Run the test, and it should **fail** because the `register()` method and `isStudentRegistered()` don't exist yet.
3. **Write the Code to Pass the Test:**
   - Implement the `register()` method and `isStudentRegistered()`.

```
public class StudentRegistry {
    private Map<String, Student> students = new HashM
ap<>();

    public void register(Student student) {
        students.put(student.getId(), student);
    }

    public boolean isStudentRegistered(String student
Id) {
        return students.containsKey(studentId);
    }
}
```

4. **Run the Test Again:**
   - The test should now **pass**. You can now move to the next feature.

▼ Step 4: Handle Edge Cases for Student Registration

1. **Write a Test for Duplicate Registration:**
   - Ensure that trying to register the same student twice throws an exception.

```
@Test
public void shouldThrowExceptionForDuplicateRegistratior
    StudentRegistry registry = new StudentRegistry();
    Student student = new Student("1", "John Doe");
```

```
    registry.register(student);

    assertThrows(StudentAlreadyRegisteredException.class
}
```

2. **Run the Test (Failing)**:
    - It will fail as there is no check for duplicate students yet.
3. **Implement Code to Pass the Test:**
    - Update the `register()` method to check for duplicates.

```
public void register(Student student) throws StudentAlre
    if (students.containsKey(student.getId())) {
        throw new StudentAlreadyRegisteredException("Stu
    }
    students.put(student.getId(), student);
}
```

4. **Run the Test (Passing)**:
    - Run the test again and verify that it passes.

▼ Step 5: Enrol Students in Exams

1. **Write the Test:**
    - Create a test to enroll a student in an exam.

```
@Test
public void shouldEnrollStudentInExam() {
    ExamRegistry examRegistry = new ExamRegistry();
    Student student = new Student("1", "John Doe");
    Exam exam = new Exam("Math", "MATH101");

    examRegistry.enroll(student, exam);

    assertTrue(examRegistry.isEnrolled(student, exa
```

```
m));
}
```

2. **Implement the `enrol()` Method**:
    - Implement functionality in `ExamRegistry` to enroll students and verify enrollment.

```
public void enrol(Student student, Exam exam) {
    List<Student> students = examEnrollment.getOrDefault
    students.add(student);
    examEnrollment.put(exam, students);
}
```

3. **Test for Edge Case: Duplicate Enrollment**
    - Prevent students from enrolling in the same exam multiple times.

▼ Step 6: Record Exam Scores

1. **Write a Test to Record Scores**:
    - Ensure that valid scores are recorded, and invalid scores are rejected.

```
@Test
public void shouldRecordValidScore() {
    ExamRegistry examRegistry = new ExamRegistry();
    Student student = new Student("1", "John Doe");
    Exam exam = new Exam("Math", "MATH101");

    examRegistry.enroll(student, exam);
    examRegistry.recordScore(student, exam, 85);

    assertEquals(85, examRegistry.getScore(student, e
xam));
}
```

2. **Handle Exceptions for Invalid Scores**:
    - Write tests to ensure scores outside 0–100 throw exceptions.

```
@Test
public void shouldThrowExceptionForInvalidScore() {
    ExamRegistry examRegistry = new ExamRegistry();
    Student student = new Student("1", "John Doe");
    Exam exam = new Exam("Math", "MATH101");

    examRegistry.enroll(student, exam);
    assertThrows(InvalidScoreException.class, () -> e
xamRegistry.recordScore(student, exam, 150));
}
```

▼ Step 7: Run Tests in GitHub Actions

1. **Set Up CI with GitHub Actions**:
   - Write a GitHub Actions YAML file to run your tests on every push.

```
name: Java CI with Maven

on: [push, pull_request]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: Set up JDK 11
      uses: actions/setup-java@v1
      with:
        java-version: '11'
    - name: Build with Maven for Testing
      run: mvn clean test
```